Beyond Network Selection: Exploiting Access Network Heterogeneity with Named Data Networking

Klaus M. Schneider
University of Bamberg, Germany
klaus.schneider@uni-bamberg.de

Udo R. Krieger
University of Bamberg, Germany
udo.krieger@ieee.org

ABSTRACT

Today, most mobile devices are equipped with multiple wireless network interfaces, but are constrained to use only one network at the same time. In this paper, we show that using multiple access networks simultaneously can improve user-perceived QoS and cost-effectiveness. We present a system architecture that exploits the adaptive forwarding plane of Named Data Networking (CCN/NDN) and implement a system prototype based on the NDN Forwarding Daemon (NFD). More specifically, we propose a set of forwarding strategies that use fine-grained application requirements together with interfaces estimation techniques for delay, bandwidth, and packet loss. Our simulation results show that our approach can improve QoS and/or reduce access costs in many wireless scenarios.

1. INTRODUCTION

Mobile and wireless terminals like smart phones and tablet pcs are becoming increasingly popular. These devices are often in the range of multiple wireless access networks like LTE, WiFi, and Bluetooth. Since each of the these access networks differs in its characteristics, like bandwidth, latency, bit error rate, security, power consumption, and network access costs, the end-user wants to select the best available network at a given time, a notion called *Always Best Connected* (ABC) [9]. The literature around ABC [12, 19, 21] contains numerous vertical handover decision strategies and network selection algorithms that aim to choose the optimal access network and to provide application transparent (seamless) handovers. However, all of these approaches assume that a terminal only uses one access network at a given time, a constraint that requires all terminal applications to agree on a single access network.

More recent solutions use multiple access networks simultaneously with the goal of improved mobility management, bandwidth aggregation, traffic offloading, or super-linear improvement of TCP performance [18]. Notable examples include SCTP, MPTCP, and IP Flow Mobility (see our previous work [17] for a review). While these approaches solve parts of the problem, they are still limited by the host-to-host communication model of IP networks. For example, IP addresses are used both as end-point *identifiers* and device *locators* for routing, which often leads to mobility problems. Moreover, since every device needs to maintain a globally routable IP address, vertical handovers are a rather heavy weight process. Instead of solving these issues with an IP overlay technology, we propose to adapt a rising clean-slate architecture.

The increasing share of content distribution among total Internet traffic has motivated the research field of *Information-Centric Networking* (ICN), which tries to shift from IP's communication model (*where*) to a content dissemination model (*what*). In this work, we use a prominent ICN architecture, called Content-Centric or Named Data Networking (CCN/NDN) [11], to better exploit mobile terminals with multiple wireless access networks.

This paper makes the following contributions. In Section 2, we argue that the information-centric model provides a better abstraction for exploiting multiple access networks than the host-centric model: devices no longer have to select a single access network, but can distribute smaller data units among a pool of available interfaces. We lay out our system requirements (Section 3) and architecture, which uses application profiles, interface estimation techniques, and NDN forwarding strategies (Section 4). Lastly, in Section 5, we describe our prototype implementation and simulation results of specific interface estimators and forwarding strategies.

2. THE NDN ARCHITECTURE

NDN has two packet types: 1) *interest* packets that request data, contain a hierarchical content name, and are routed towards a permanent storage location (*repository*), and 2) *data* packets that follow the path of the interests like Hansel and Gretel (intended to) follow their trail of breadcrumbs. Data packets may be cached on and retrieved from any intermediate router.

NDN routers consists of three parts: 1) The *Content Store* that acts as a temporary cache, 2) the *Pending Interest Table* (PIT) that stores a mapping of a content name to a set of requesting inbound faces, and 3) the *Forwarding Information Base* (FIB) that stores a mapping from content name prefixes to a set of outgoing interfaces (instead of exactly one in IP routers). If the FIB contains multiple alternative paths for a prefix, the NDN *strategy layer* decides where to forward incoming interest packets. In our scenario of multihomed end-devices (Figure 1) the NDN architecture shows the following differences to the IP model:

1) Content identifiers instead of host identifiers. NDN's unique forwarding layer removes all network layer host identifiers, implicitly splitting them from host locators. Since NDN hosts do not need to maintain a globally routable IP address, consumer mobility becomes automatic and seamless. Producer mobility is harder to achieve, but has been frequently discussed in the past [13, 16, 20, 24]. The NDN strategy layer is not constrained to select a single network for the whole terminal, but can decide on a set of outgoing interfaces for each incoming interest packet. This *interest distribution* is more

ICN'15, September 30–October 2, 2015, San Francisco, CA, USA.
 ISBN 978-1-4503-3855-4/15/09 ...$15.00.
DOI: http://dx.doi.org/10.1145/2810156.2810164.

lightweight than vertical handovers in IP, as content sources do not need to know the location (i.e., the IP address) of the requester: interest packets can be sent unmodified on any available interface; IP packets need a different source address.

2) Loop-free multipath routing. Since interest packets carry a nonce and data packets follow the exact path of the interests, neither of the two NDN packets can loop. This loop-freeness allows the NDN strategy layer to send out one incoming interest simultaneously on multiple outgoing interfaces, a feature that can be used to actively probe links and to improve the overall QoS.

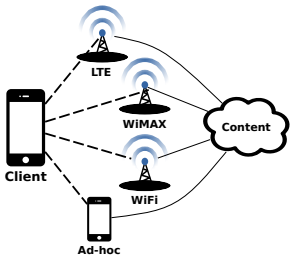


Figure 1: Multihomed Terminal Scenario

3) Connectionless, yet stateful forwarding. NDN's stateful forwarding mechanism allows the forwarding strategy to store performance information about each available interface. This information can later be used to react fast and adaptively to changes in link quality. For example, NDN forwarding can detect and mitigate link failures in the order of milliseconds [23]. Moreover, stateful forwarding allows the strategy to aggregate the available bandwidth, to handle delay tolerant traffic, to improve handover performance, and to distribute data in a P2P fashion [17].

4) Support of conversational traffic. In addition to content traffic, NDN also supports conversational traffic like voice calls, video conferences, or email. Example applications include Voice-over-CCN [10], audio conferencing [25], and P2P video streaming [7].

3. SYSTEM REQUIREMENTS

Some goals of multihomed terminals, like seamless handovers, are intrinsically supported by NDN; others require additional effort. Below we list the requirements of our system that exceed current NDN functionality. These stem from a number of common observations about access network and traffic characteristics:

1) Applications differ in their requirements to the network. Some applications, like file downloads, benefit more from increased throughput, others, like voice calls, more from reduced packet loss or delay. Moreover, the function of user-perceived Quality of Experience (QoE) to network-level QoS differs for each application type (see Figure 12 in Section 5.3). For example, the voice quality of VoIP applications that use adaptive encoding will reach a threshold (in the order of 10's of kbit/s) at which coding with a higher bandwidth no longer makes any noticeable difference to the user. Such a threshold can also be seen with file transfers (a user does not care much if a file takes 0.1s or 1s to download), but there it is less clearly defined and also depends on the file size, as large files benefit more from high bandwidth. The same applies to video codecs, where the threshold depends on factors like the screen size and screen resolution.

2) Applications differ in their relative importance to the user; most users perceive the reliability of a bank transfer as more important than that of a NTP time synchronization request. Thus, they may want to prioritize these important traffic flows over others.

3) Access networks differ in their link characteristics such as bandwidth, round-trip delay, bit error rate, and availability. For example, WiFi has a higher maximal throughput and higher power consumption than Bluetooth. Moreover, the signal quality of a wireless network strongly affects the QoS: devices close to their access point typically perform better than those far away.

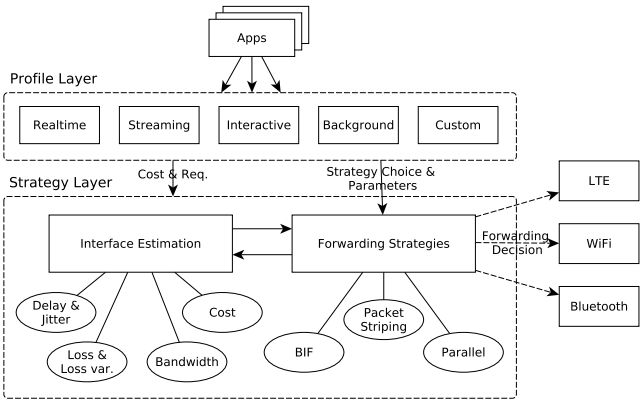


Figure 2: High-Level System Architecture

4) Access networks differ in cost factors such as monetary costs, bandwidth limits, and power consumption. Users have a different cost tolerance for each application; thus, they may want to offload low-importance traffic to less expensive networks, while keeping high-importance traffic on expensive, but high quality networks. For example, many mobile applications prefer to transfer certain high-bandwidth traffic like software updates only while connected to (inexpensive) WiFi networks. These considerations lead to the following system requirements:

1) Cost- and energy-awareness. The system should allow different notions of costs; for example, one to model energy consumption and one to model a cellular traffic limit. The system should allow to automatically offload lower-importance traffic to lower-cost interfaces.

2) Reacting to changes in link quality. The system should detect and handle both QoS deterioration and recovery. This adaptation to QoS levels should work more fine-granular than recognizing only complete link failure and recovery.

3) Exploiting multiple network interfaces. The system should use multiple network interfaces in parallel to improve overall QoS.

4) Restricting changes to the terminal. The system should not require to change devices inside the network such as access or backbone routers (other than their support of the NDN protocol stack). Limiting changes to the terminal eases deployment and allows the terminal to opportunistically exploit new access technologies.

In a related work, Detti et al. [6] proposed a cooperative video streaming application that offloads traffic from cellular networks to local peers connected by a WiFi ad-hoc network. One key difference to our approach is that they use routing to locally address content: peers add known content chunks to their FIB and the consumer application performs a longest prefix match on the chunk name, which returns only the face of the local peer. In contrast, our system relies mainly on forwarding: we assume that the FIB contains multiple interfaces per prefix (prefixes can be more coarse-grained than chunk-level) and let the forwarding strategy decide which one to use.

4. SYSTEM ARCHITECTURE & DESIGN

Our system architecture uses three components that work on and above the NDN strategy layer (Figure 2): 1) *application profiles*, 2) *estimation of interface characteristics*, and 3) *forwarding strategies.*

4.1 Application Profiles

Applications have different network requirements which we model as specific thresholds of maximal delay, maximal packet loss, maximal cost, and minimal bandwidth. These requirements are meaning-

ful to the forwarding strategy, but not necessarily to the application developer or even end-user. To fill this gap we use a *profile layer:* application developers can select high-level profiles that contain low-level requirements and hide their complexity. We predefine four of these profiles to model common traffic classes [1]:

1) Conversational real-time services (RTS) include highly interactive applications like voice calls, video conferences, and real-time games. They are typically important to the user, require a round-trip delay below a strict threshold (less than a few 100 ms), and often tolerate packet losses up to a certain degree. Some RTS, like voice calls and certain types of games, require only moderate bandwidth; others, like video conferences, require a much higher amount.

2) Interactive services, like web browsing or email, are often delay-sensitive, that is, the RTT should not exceed a few seconds. These services often require reliable end-to-end transport and low-to-moderate bandwidth capacity.

3) Streaming services, like stored audio/video streaming and file transfers, are similar to real-time services, but with a relaxed delay requirement: they tolerate start-up delays up to a few seconds. However, they might be sensitive to delay variation (jitter). Streaming applications have a limited playout buffer (known as *jitter buffer*) and have to discard packets that arrive too late, that is, outside the buffer. These services often consume large amounts of bandwidth and, thus are a prime candidate for offloading to less expensive networks like WiFi.

4) Background applications include server-to-server email transfer and SMS-like messaging applications. These applications typically tolerate delays above 10 seconds, which allows them to perform reasonably well on lower cost network interfaces.

In addition to QoS elements, an application profile also contains one or multiple cost factors to model, for example, provider cost and energy consumption. These costs are usually a function of the bandwidth that an application consumes; thus, the cost-level can be specified relative to the number of requested packets (e.g., 5 cost units per data packet) or to the size of the consumed traffic. Another use of the cost factor is to model privacy and security requirements: insecure links can receive a higher security cost level and applications can specify a corresponding cost threshold. The semantics and usage conventions of cost attributes should be described by an API of each individual forwarding strategy.

If the application developer has deeper knowledge about network-level QoS, he can create a custom application profile. Moreover, if the application does not specify its profile, the profile layer can try to guess the traffic class by observing traffic patterns or user behavior. Lastly, a single application may define multiple traffic flows with different QoS profiles. For example, an email client may want to treat a user-triggered fetch as interactive service and treat regular polling as background traffic.

4.2 Estimation of Interface Characteristics

To forward packets intelligently, the strategy layer needs to know the characteristics of all available interfaces. In particular, our system considers the interface metrics of delay, jitter, packet loss, loss variation, bandwidth, and different types of cost (see Figure 2). These characteristics cannot be measured with perfect precision, but can often be estimated accurately enough to base forwarding decisions on them. This estimation is done by a number of interface estimator modules, which share the following design principles.

Interface estimation has to work on an appropriate abstraction level. Typically, wireless networks differ in physical characteristics like bit error rate (BER) and low-layer mechanisms like the modulation technique, FEC codes, and retransmissions. Since the presented system should work with any available access technology, these details are abstracted away: the following estimation techniques measure network-layer performance and treat lower-layer behavior as a black box. Most of them fall into two categories: 1) passive traffic monitoring and 2) active probing.

Passive monitoring observes the "natural" traffic flow of applications to estimate interface characteristics; thereby, it avoids additional traffic overhead, but also misses some features of active probing. For example, passive probing cannot detect *link recovery*, that is, a QoS improvement of a path which is not among the current working paths; it is hard to determine the performance of an interface that does not send or receive data.

In contrast, *active probing* can probe interfaces other than the current working path to find recovered or "better" interfaces. Since active probing introduces overhead, the strategy has to adjust the probing frequency to trade-off lower traffic overhead with more precise detection of link quality.

In addition, some metrics can be retrieved from external input like hardware or network provider information. These depend on the specific metric, like bandwidth estimation, and are discussed in the next section.

Interface Estimators need to report their information to the forwarding strategy. We use a *moving average* calculation to smooth out noise and to reduce the undesirable effects of random variation. Moving averages can be computed in two ways: An *exponential moving average* (EMA) computes the average by multiplying a smoothing factor α with the current value Y_t and adding it to $(1-\alpha)$ times the previous values:

$$EMA_t = \alpha * Y_t + (1-\alpha) * EMA_{t-1} \tag{1}$$

This method uses all older values of Y_t, but decreases their importance exponentially. In contrast, a *simple moving average* (SMA) keeps a number of n previous values and computes the arithmetic mean over them:

$$SMA = \frac{Y_t + Y_{t-1} + \ldots + Y_{t-(n-1)}}{n} \tag{2}$$

Every time a new value is added, the oldest value drops out of the computation. A SMA can be computed over the last *n packets* or the last *m time units*, which results in a variable number of n packets, depending on how many packets were sent or received during that time period. We report specific results about the moving average calculation of the loss estimator in Section 5.

4.3 Forwarding Strategies

The main functionality of our system is implemented by NFD forwarding strategies. A forwarding strategy uses information from FIB entries, from application requirements, and from interface estimation to decide where to send on incoming interest packets. We assume that the FIB routes are correct, that is, interfaces from the FIB return content unless there is an unexpected failure. The forwarding strategies in our system are not required to probe paths outside the FIB to find off-path cache copies, but may optionally do so. Each strategy class defines an API for the profile layer which contains the strategy's set of parameters together with their semantics. Instances of these strategy classes contain specific parameter values, are bound to a name-prefix, and are looked up with a longest-prefix match. Moreover, the strategy informs the interface estimators about sent and received packets and performs active probing measurements.

Forwarding strategies can also perform interest retransmissions. These strategy layer retransmissions can be useful, because they can access more context information (e.g., about multiple paths) than higher layers. For example, retransmitted packets can be sent on a different (e.g., more costly) interface than before to increase the

chance of data retrieval. A common practice is to perform strategy retransmissions if the application requests a given packet again. Another option for a router is to retransmit interests after a timeout, independently of the application. However, Abu et al. [2] argue that the latter approach is undesirable, because it increases the network load without reducing the number of PIT entries.

We categorize forwarding strategies into three classes [17]:

1) Best Interface First (BIF) strategies send out interest packets on a pre-determined "best" interface and may choose others for retransmitted interests. These strategies change their interface ranking dynamically according to variation in the QoS of the employed links (as reported by the interface estimators). BIF is the default and probably the preferred choice for most traffic classes; however, special cases may benefit from one of the other categories.

2) Packet Striping strategies can split a single application flow on different network interfaces to aggregate their bandwidth, allowing the user to run applications whose bandwidth requirement cannot be satisfied by a single link. However, since the available bandwidth per access technology varies greatly, this approach seems rarely useful on real-world terminals [18].

3) Parallel strategies send out interest packets redundantly to minimize content response time and packet loss in a trade-off against higher costs. The strategy uses the first returning data packet and discards the later arriving ones, thus reducing the overall content response time to the minimum of the response times of all involved interfaces (IF):

$$RTT_{total} = min\{RTT_i \mid i \in IF\} \tag{3}$$

The impact on packet loss depends on the loss distribution of all involved links. In the worst case, losses are totally correlated and the overall loss rate is the minimum of the loss rates on the links (p_i):

$$p_{loss} = min\{p_i \mid i \in IF\} \tag{4}$$

If the losses of different links are independent, the total loss rate is the product of the individual loss rates:

$$p_{loss} = \prod_{i \in IF} p_i \tag{5}$$

Although unlikely, losses may be negatively correlated, that is, a loss on one channel makes it more likely to retrieve data on another channel. In the best case (corr = -1) the total loss rate is much lower than that of a single link:

$$p_{loss} = max\{0, 1 - \sum_{i \in IF} (1 - p_i)\} \tag{6}$$

The question arises to which loss model most accurately approximates real-world wireless packet losses. When using technologies with different access ranges, like WWAN and WLAN, a mobile terminal may well experience independent loss characteristics. However, if technologies are similar, like a connection to two WiFi access points, correlated losses are more likely. Since parallel strategies produce a large overhead they are most useful for traffic with high importance, low total bandwidth, and strong loss and/or delay requirements, that is, for real time communication services. We provide the design and initial evaluation of one of such strategies in Section 5.3.

5. PROTOTYPE IMPLEMENTATION & PERFORMANCE EVALUATION

Below we describe a proof-of-concept implementation of our system's strategy layer. We have implemented the system based on the NFD platform, because of NFD's active development community and modular C++ code; however, with a few adjustments, one can also implement the system in CCNx [4]. We have released the source code of our implementation for experimentation at [8].

5.1 Shared Elements

Before diving into the strategy specifics, we describe a number of shared components:

Interface Probing.

The following strategies perform *active probing*, that is, they send out interest packets on paths other than the current working path to detect changes in interface performance. Since probing provides no direct performance benefit, the strategy has to trade-off the accuracy of interface estimation with the network overhead of probing packets. To control the probing effort, we follow Yi et al.'s [22] suggestion to split the probing into two functions: `bool probingDue()` and `void probeInterfaces()`.

The function probingDue() decides *if* other interfaces are probed at a given time point. We currently use two versions of probingDue: 1) one that probes interfaces (i.e., returns TRUE) every x packets, and 2) one that probes for all incoming packets during periodic time periods. Other options are to probe every y seconds, probe both every x packets and every y seconds, or use more complicated algorithms. When probingDue() returns TRUE, the strategy runs the function probeInterfaces() that decides *which* interfaces are probed. Currently, we probe all faces except the current working face (which receives the packet anyway); other options are to probe a random face and to probe faces in round robin.

Probing packets use a different nonce to avoid being dropped by the loop detection of an intermediate router. For example, in the topology of Figure 3, if an interest is sent with the same nonce both on face 256 and, for probing, on face 257, the router `Backbone` thinks that the later arriving one is looped and drops it. Consequently, one of the faces does not return a data packet, distorting its performance estimation.

Strategy parameters.

The profile layer needs a mechanism to pass parameters to the selected strategy instance. Since NFD does not natively support strategy parameters, we encode them as part of the strategy name with the syntax

$$\texttt{/strategy-name/version/}p_1\texttt{=}v_1, \ldots, p_n\texttt{=}v_n \tag{7}$$

where p_i denote parameter names and v_i denote parameter values. Some parameters have a lower value vl_i and an upper value vu_i, which we connect with a dash: p_i=vl_i–vu_i

Like the strategy instance itself, the parameters are saved per name-prefix and looked up in a longest-prefix match (LPM). If the LPM does not return any entries, the strategy uses a set of pre-defined default parameters.

We currently pass the interface cost through NDN's routing cost value. If a strategy needs more than one cost attribute, this can also be done via parameters.

Measurement Information.

A forwarding strategy has to store performance measurements of the available links. For this purpose, it creates instances of the interface estimator classes described below. Some of this information, like interface cost, is the same for all applications flows and is stored per-face; other information differs for each application and must be stored per-name-prefix-per-face. For example, the round-trip delay of two flows can differ strongly if they share the same network interface, but use different paths in the backbone network.

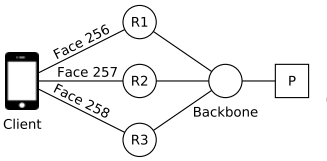


Figure 3: Topology

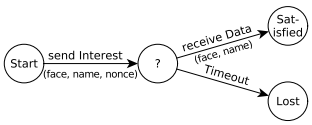


Figure 4: Loss State

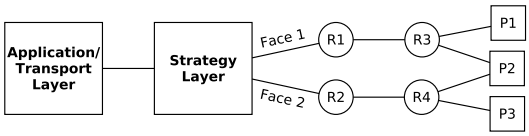


Figure 5: Congestion Control Example

Our implementation currently saves this per-name-prefix information at one level upward of interest names, which by convention end with a sequence number. Thus, if an application flow requests the packets /prefix/A/seq1 to /prefix/A/seq100 they share the same measurement information; another flow that requests /prefix/B/seqXY maintains separate performance measurements. If storing measurements one level up creates too much overhead, one can aggregate measurement information further, like storing them two levels up. Measurement information is looked up in a longest-prefix match and may default to per-face information for new applications.

Performance Evaluation.

We have performed a simulation in ndnSIM 2.0 [14] to point out certain characteristics of our implementation. The simulation topology (Figure 3) models a wireless terminal with multiple access networks and is shared by all evaluation scenarios; however, the consumer applications and link characteristics differ, and are described in detail in the specific scenarios below.

Congestion Control and Caching Effects.

Congestion control in NDN is still an open research topic. Typically, NDN consumer applications use a congestion window with a TCP-like Additive Increase Multiplicative Decrease (AIMD) mechanism. The window is increased on returning data and decreased on out-of-order packets, timeouts, or delay variation [5].

Congestion control at the consumer is difficult, because of NDN's multipath strategy layer and its use of in-network caching: both are transparent to the application and create a large variation in link QoS. For example, a forwarding strategy can transparently change the outgoing face, leaving the consumer's congestion window maladapted to the new path characteristics. This adverse effect of path variability can be reduced by giving the congestion-control algorithm information about the underlying network. A first step is to use the *per-face* information of the strategy layer: all faces in the FIB (F1 and F2 in the example of Figure 5) maintain their own congestion window and measurement information, a change that improves the performance when the strategy switches between faces, but ignores path differences that lie beyond the first hop. For example, Face 1 in Figure 5 can be served by two content producers, introducing variability when the backbone router C3 changes between them. To address these path differences, one can keep *per-route* information [3], that is, one can discern between non-disjoint paths towards permanent content producers (like the 4 non-disjoint paths to the 3 content producers in Figure 5).

This per-route information allows a more fine-grained congestion control, but requires to extend NDN packets with route labels and

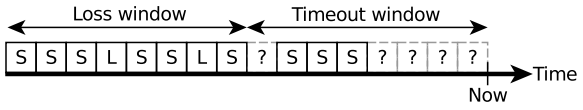


Figure 6: Packet Loss Map

still leaves the problem of high path variability due to in-network caching. For example, a packet answered by a cache (cache hit) can have a much lower rtt than one answered by a producer (cache miss), a difference that may impair delay-based mechanisms for retransmission timeouts and congestion control. The effect of this variability is different for each of the interface estimators (delay, packet loss, and bandwidth); bandwidth estimation, for example, may be more constrained by a bottleneck link than by the cache hit ratio. One option to avoid the caching variability is to keep *per-cache* congestion windows in addition to per-producer information (8 routes to 7 caches/producers in Figure 5). However, the number of possible caches is unknown and possibly much larger than that of content providers, which could lead to an unacceptable memory overhead.

More coarse-grained solutions avoid this overhead and some argue that these are still effective in reducing congestion; examples include using congestion-control without per-path information [5] and using one shared congestion window for all routes together with per-route measurement information [3]. Another rationale for coarse-grained congestion control is that, in wireless networks, the access links often have a much higher impact on QoS than the backbone network. In this case, one can safely use per-face congestion control and ignore caching effects.

We have experimented with the *ConsumerWindow* application of ndnSIM; however, its current design is unsuitable for our scenario, since it resets the congestion window on each timeout, leading to unstable measurement results. Thus, we leave congestion control measurements for future work and, in the following experiments, use a consumer with a constant bit rate.

5.2 Interface Estimators

The current implementation estimates interface performance according to the three main QoS metrics: delay, packet loss, and bandwidth.

Delay Estimator.

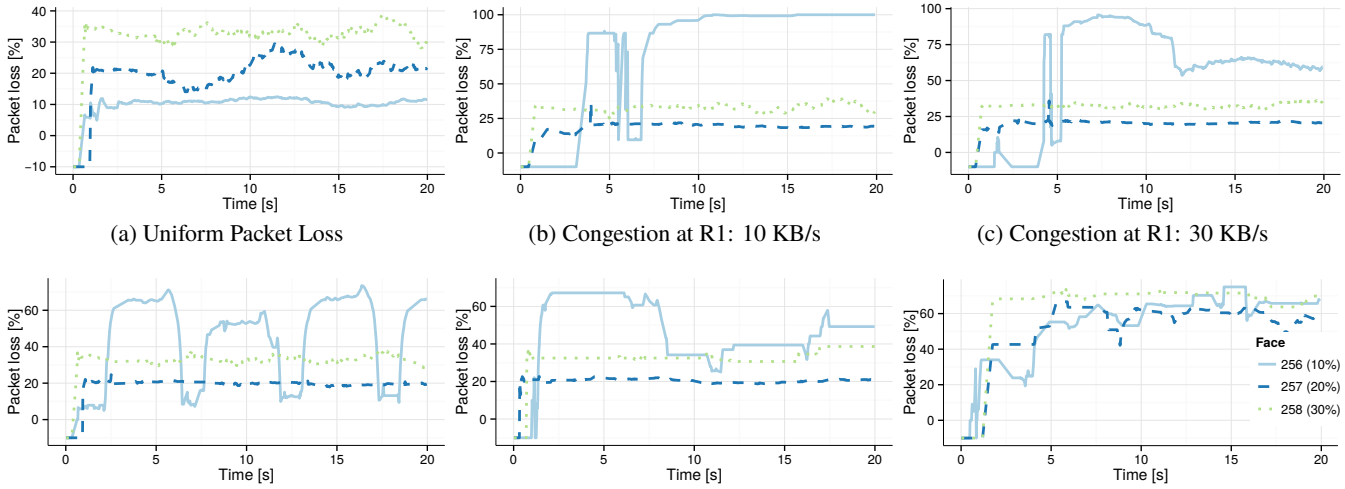
The implemented delay estimator is based on the RttEstimator of the NFD prototype, which itself is based on the NS-3 simulator. On every returning data packet, it adds the new rtt value to the old ones using an exponential moving average ($\alpha = 0.1$). We use two output values of the delay estimator: 1) the mean round-trip time (rtt) and 2) the *request timeout* (rto), that is, the mean rtt plus 4 times the variance.

Loss Estimator.

Since there is no existing loss estimator in NFD, we present our own design. Unfortunately, the loss estimator cannot use an exponential moving average, because there are no discrete values like "x ms" for packet loss. Instead, an interest packet can have one of three loss states (see Figure 4):

- *Satisfied:* A data packet has returned for that interest
- *Unknown:* A data packet may return in the future
- *Lost:* No data packet has returned before the timeout

We use these states to calculate a simple moving average over the last n time units.



(a) Uniform Packet Loss (b) Congestion at R1: 10 KB/s (c) Congestion at R1: 30 KB/s

(d) Congestion at R1: 100 KB/s (e) Congestion at R1: 100 KB/s, bursty probing (f) Congestion at Backbone: 200 KB/s

Figure 7: Performance of the Loss Estimator

A trivial solution is to simply count the number of sent interest and received data packets over a given time period and to calculate the loss percentage as follows:

$$p_{loss} = \frac{I_{sent} - D_{rec}}{I_{sent}} \tag{8}$$

However, since data arrives later than interests are sent out, received data packets may count for a different interval than their corresponding interests, which causes large fluctuations in loss estimates; an interval that contains more data than interest packets even results in a negative loss estimate.

To estimate packet loss more accurately, the estimator must maintain a unique association between sent interest packets and returning data packets. Interest and data packets can be matched according to the content name and the respective interface (see Figure 4); however, the consumer application can retransmit lost packets, which results in multiple sent interests per received data packet. We assume that the strategy layer can detect these retransmissions, for example, by recognizing interests with a duplicate content name or a duplicate nonce, coming from the local application face.

Our loss estimator stores each sent interest in a key/value map `lossMap <time stamp, content name>`. For each returning data packet, it clears the content name to mark a *satisfied* packet. Entries that still have their content name are considered *unknown* before the timeout and *lost* after it (see Figure 6). The timeout is currently set based on delay estimates (the mean rtt plus 4 times the variance); another option is to use the interest lifetime (i.e., the time that an application still considers the data to be useful). When the loss estimator detects a retransmitted packet, it marks the earlier packet as *lost* and ignores all later arriving duplicates (implemented by changing the content name inside the loss map to "lost").

Finally, we calculate the loss percentage as ratio of lost to total packets during the *loss window*, a sliding window that starts after the timeout:

$$p_{loss} = \frac{N_{LOST}}{N_{LOST} + N_{SATISFIED}} \tag{9}$$

If the current loss window is empty, the loss estimator returns a negative value (currently -10%). The forwarding strategy can decide how to handle these missing loss estimates; in the following, we treat them like a loss-free channel.

We have measured the performance of the Loss Estimator in ndnSIM with the *ConsumerCBR* application that sends 500 interest packets per second and retransmits lost ones after a delay-based timeout. The paths are modeled with a uniform loss distribution of 10%, 20%, and 30% and respective delays of 200 ms, 300 ms, and 500 ms (using the topology of Figure 3). The strategy uses a loss window of 4 seconds, sends interests on the lowest-loss face, and probes all other faces every 10th packet.

We observed that, after the individual timeout, the loss estimator is very *accurate*, that is, the mean of the reported loss value matches the actual packet loss (Figure 7a). We noticed that the loss estimation is barely affected by static differences in link delay; however, delay-based timeouts may overestimate packet loss if the delay estimation is inaccurate. This problem can be avoided by using the interest lifetime (default 2s) as timeout, but that introduces a static and high delay for the loss estimator to react to changes.

The loss estimator is more *precise* for the working path, that is, it has a low variance of reported values, as the variance depends on the traffic load on the channel, the probing frequency, and the size of the loss window. To improve the precision, one can increase the probing frequency, at a higher overhead, or increase the duration of the loss window, at a higher delay until the strategy reacts to changes in packet loss.

In addition to uniform random loss, we consider the effect of congestion-based packet losses on loss estimation and strategy behavior (Figure 7b - 7f). We have limited the bandwidth of face 256 (10% loss) below the requested data load, thereby producing additional packet drops at the queue of an intermediate router (R1 in Figure 3). If the bandwidth at R1 is limited to a degree that a large percentage of the probing packets are lost, the loss estimator reports a high loss value, allowing the strategy to avoid the congested path (Figure 7b and 7c). However, if the congested link still allows the probing traffic to go through, an oscillation pattern arises (Figure 7d): First, face 256 is selected as working face, because it has the lowest packet loss (10%). After a while, the load on the path leads to increasing packet losses (and loss estimates) at R1, so the working path changes to face 257. Path 256 now only has to transfer the lower probing load, recovers, and the pattern repeats itself. A trivial solution for this problem is to increase the amount of probing, but that also increases traffic overhead.

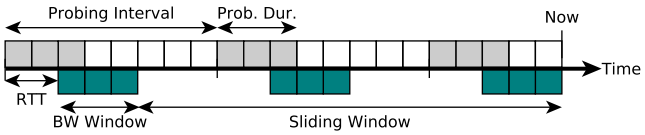
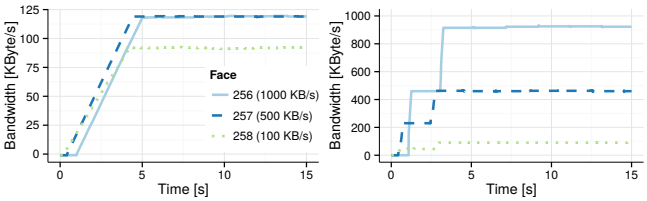


Figure 8: Bandwidth – Burst Estimation



(a) Passive Estimation (b) Burst Estimation

Figure 9: Performance of the Bandwidth Estimator

An approach that avoids additional overhead is to use *bursty packet probing:* instead of probing every 10th packet, we now probe all packets during bursts of 400 ms every 4 seconds (the probing interval must be shorter or equal to the loss window to ensure that at least one probing burst falls inside the window). In this case, a part of the probing load is lost due to congestion and the loss estimation stays at a level that discourages the use of the congested path (Figure 7e).

Another scenario is congestion at a shared router in the backbone network. In this case, the congestion-based loss often dwarfs the random loss at the local links, so that the loss estimator (with both steady and bursty probing) detects an equal congestion on each path (Figure 7f). To prevent the strategy from randomly switching between them, one can use a *hysteresis*, that is, a policy to only change the working path if another path has a loss percentage which is at least x percent lower than the current one. We discuss two specific hysteresis implementations in the next section.

Bandwidth Estimator.

A strategy could probe the bandwidth the same way as loss and delay, that is, it could send out probing packets and observe the throughput of the returning data; however, that would require to fill the capacity of all non-working paths, creating an unacceptable traffic overhead. One option to avoid that overhead is to use *passive bandwidth estimation*, that is, to observe the traffic that naturally flows over each path. However, this method only returns accurate bandwidth estimations of the current working path (or multiple working paths); for other paths, it returns the throughput of the probing traffic plus the traffic of other applications, often a much too small value.

We use another bandwidth estimator, called *burst estimator*, that uses short bursts with high traffic load (flooding) and measures the resulting throughput (see Figure 8). The burst estimator performs regular probing with a fixed interval length and probing duration, such as probing every 3 seconds for 100 ms. It measures the returning data traffic that falls inside periodic bandwidth windows which follow the probing window with a delay of one rtt, that is, the time the data of the probes is expected to return. Moreover, it calculates a simple moving average over a sliding window, which must be a multiple of the probing interval to assure that it always contains the same number of bandwidth windows. The burst estimator extrapolates the throughput during the bandwidth windows to the whole sliding window:

$$BW = \frac{\sum DataInBwWindow}{slidingWindowSize} * \frac{probingInterval}{probingDuration} \quad (10)$$

We have compared the two bandwidth estimators in the following measurement scenario: An application creates a constant data stream of 1200 KB/s, but the strategy always chooses link 258, which is limited to 100 KB/s. The other two links have a bandwidth of 500 KB/s and 200 KB/s, respectively, and are probed every 10th packet. The link delays are 1000 ms (link 256), 500 ms (link 257), and 200 ms (link 258).

The passive bandwidth estimator returns accurate values for the working path, but vastly underestimates the other two, which only receive an estimate of 120 KB/s, exactly the throughput of the probing traffic (see Figure 9a).

The burst estimator probes all paths every 2 seconds for 200 ms and uses a sliding window of 4 seconds. It estimates the bandwidth of the non-working path much more accurately (Figure 9b), without increasing the traffic overhead. Moreover, it is not influenced by static differences in link delay, as it takes the delay estimates into account; however, the estimates may be skewed by delay variability, that is, when the actual path delay differs from delay estimates.

In addition to these two estimators, there is a large literature of bandwidth estimation techniques for IP networks [15]. Future work could evaluate if and how those techniques can be adapted to NDN.

5.3 Forwarding Strategies

According to Occam's Razor, the simplest solution to a given problem should be preferred; more complex systems need to be justified by added functionality. Therefore, we describe three forwarding strategies that range from low to higher complexity, and show their merits in different evaluation scenarios.

Lowest Cost Strategy.

The Lowest Cost Strategy (LCS) takes three parameters which specify application requirements: maximal packet loss (maxloss), maximal delay (maxdelay) and minimal bandwidth (minbw). It uses an ordinal ranking of costs and sends out packets to the interface with the lowest cost that satisfies all requirements. The requirement values are considered as hard thresholds, that is, if the profile sets a maximal delay of 100 ms, the strategy considers a link with a delay of 101 ms as inappropriate and, if possible, chooses one with a lower delay. When none of the interfaces can satisfy all requirements, the strategy uses one *priority attribute* to decide where to forward packets; for example, if the priority is set to `delay` and no face satisfies all requirements, the strategy chooses the lowest-cost face that satisfies the delay requirement and, if this is not possible, the face with the lowest delay.

Even with this simple design, the Lowest Cost Strategy has a number of advantages over the currently available forwarding strategies. First, it can choose the best interface according to application requirements. Second, it can detect a QoS degradation of the current working path and switch to a better path. Third, it can detect fine-grained path recovery, like a reduction in packet loss from 5% to 2%, and switch to the recovered face if desired. We investigate how far the LCS achieves these goals in the following performance evaluation.

Figure 10 shows the functionality of the Lowest Cost Strategy in a prototypical test scenario. The client has three paths which are modeled with the following characteristics: Face 256 has a RTT of 300 ms, 5% packet loss, and the highest cost. Face 257 has 400 ms delay, also 5% packet loss, and medium cost. Face 258 has 200 ms delay, 30% packet loss, and the lowest cost. The strategy is

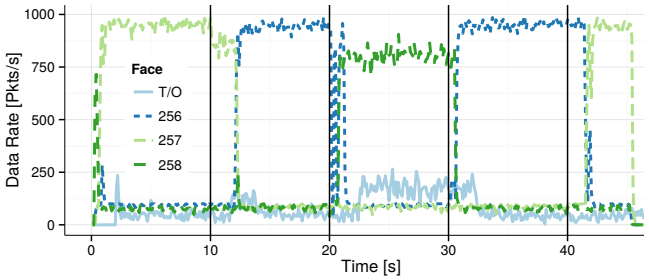


Figure 10: Behavior of the Lowest Cost Strategy in the Case of Link Deterioration and Recovery

configured with the parameters maxdelay=500[ms],maxloss=10[%] and the priority attribute is set to "delay". The interest lifetime is 2 seconds and the loss estimator uses a delay-based timeout together with a loss window of 3 seconds. Moreover, we set up four events at different time points of the measurement run:

- 10 sec: Loss deterioration of path 257 to 15%.
- 20 sec: Delay deterioration of path 256 to 700 ms.
- 30 sec: Delay recovery of path 256 back to 300 ms.
- 40 sec: Loss recovery of path 257 back to 5%.

We observed the following behavior: After a short phase where no loss estimation is available yet, the strategy selects face 257 as the one with the lowest cost that satisfies all requirements. After the loss deterioration, the strategy sets the working face to 256 and after the delay deterioration to 258. Similarly, after the loss and delay recovery, the strategy selects the previous faces again.

We observe that the strategy switches paths faster after delay changes than after loss changes, caused by the differences in the two interface estimators: the EMA of the delay estimator adjusts faster to changes than the SMA of the loss estimator. The timeout face (T/O) shows the application-level timeouts (retransmissions are disabled) which follow the loss characteristic of the selected interface with a delay of the interest lifetime, that is, 2 seconds. The timeout rate increases at the start of the loss deterioration before the face changes to 256 (at 10+2 seconds) and whenever face 258 is selected. The application-level delay, which is not plotted, behaved as expected: the RTT was between 200 and 700 milliseconds, depending on the selected path.

In a second evaluation (Figure 11), we want to show how precise the selection of a path works around the parameter threshold. We set up two paths: one with low cost and a delay value of 200 ms without variance; another one with a higher cost, but lower delay (the exact value does not matter). Figure 11a shows the amount of traffic that flows on the path with 200 ms. For threshold values above 200 ms, the strategy sends all traffic except the 10% probing overhead on the expected path. However, for values below 200 ms, some traffic is sent over the path with a delay that is higher than specified. This behavior can be eliminated by using a hysteresis: all paths that are not the current working path must be at least be x (currently 20) percent better than the given requirement to become the working path; for example, if the delay requirements is 200 ms, non-working paths need to be below 180 ms. For the non-variable delay measurement, this creates a clear switching point at 200 ms (Figure 11a).

We repeated the measurement with a mean packet loss of 20% and high loss variance (loss estimation report in Figure 11c). In this case, the hysteresis no longer creates a clear switching point (Figure 11b) and moderately biases the choice towards the face with lower loss and higher cost. However, with the hysteresis, the strategy stays much longer on the selected interface (either lower-loss or lower-cost) as seen by the number of switches of the working face (Figure 11d).

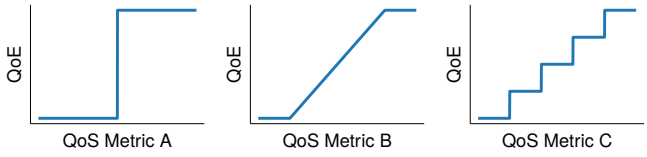


Figure 12: Generic Relationship Between Network-level QoS and User-perceived QoE

MADM Strategy.

One limitation of the Lowest Cost Strategy is its implicit and imprecise notion of costs. To allow multiple and more fine-grained cost attributes, we designed a strategy based on Multiple Attribute Decision Making (MADM). In addition to cost-awareness, the MADM strategy takes the relationship between application-level QoE and network-level QoS into account (see Figure 12 and the discussion in Section 3). Most applications show a linear (B) or step wise (C) relationship rather than a hard threshold (A). To model metric B and to approximate metric C, the MADM strategy uses two threshold values: an *upper threshold* (max_i), which achieves the maximal possible QoE for the application, and a *lower threshold* (min_i), which is the bare minimum QoS requirement, that is, anything worse is unacceptable. For example, an adaptive video streaming application could set its thresholds to $min_{bw} = 100\ Kbit/s$ (lowest video quality) and $max_{bw} = 5\ Mbit/s$ (highest video quality). The attributes that lie between these thresholds are then normalized to an interval between 0 and 1. The simplest option is linear normalization, as seen in metric B of Figure 12; moreover, one can use exponential, logarithmic, or sigmoidial functions [19] to model diminishing returns of higher QoS like additional bandwidth.

One must distinguish between *upward attributes* (e.g., bandwidth) where higher values denote a better QoS and *downward attributes* (e.g., delay, cost, loss) where the opposite is the case. Considering these details, we normalize each attribute *i* with a linear function:

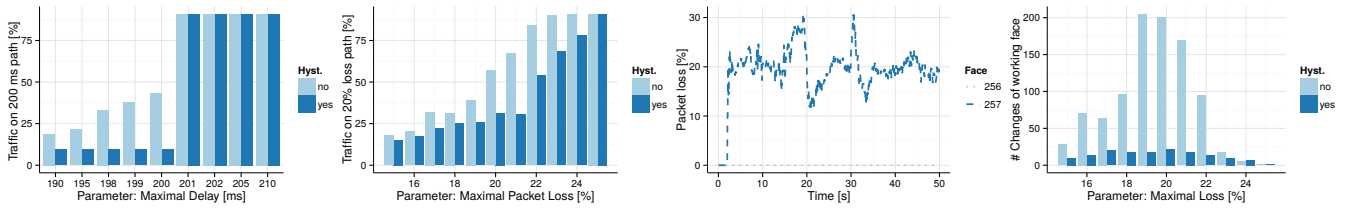
$$vu_i = \begin{cases} 0 & \text{if } x_i \leq min_i \\ \frac{x_i - min_i}{max_i - min_i} & \text{if } min_i < x_i < max_i \\ 1 & \text{if } x_i \geq max_i \end{cases} \quad (11)$$

Downward attributes invert the normalized value: $vd_i = 1 - vu_i$ For example, an application profile could set the requested minimal delay to 100 ms and the tolerated maximal delay to 300 ms. In this case, $v_{del} = 1$ for any observed delay below 100 ms, $v_{del} = 0.5$ for a delay of 200 ms, and $v_{del} = 0$ for a delay above 300 ms. To model a hard threshold, like metric A in Figure 12, the application profile can specify only one attribute value. In this case, both min and max are set to this value and the same formula applies.

These normalized values are then weighted and the strategy sends the interest packet on the interface with the highest total value. Currently, we use simple additive weighting (another option is multiplicative exponential weighting) of all attribute values v_i with all weights w_i set to 1:

$$V_{SAW} = \sum_{i \in IF} w_i * v_i \quad \forall w_i . v_i \geq 0 \quad (12)$$

In addition to attribute weighting, our current implementation makes a number of fine adjustments: First, if one attribute value is 0, the total sum of the face is set to 0. This measure prevents that



(a) Delay Threshold (b) Loss Threshold (c) Loss Estimator Output (d) Number of Switches

Figure 11: Effect of a Hysteresis on the Precision of the Lowest Cost Strategy

a path which is insufficient in one attribute (and good in others) is chosen over one that is at least acceptable in all of them. Second, to prevent unnecessary oscillation between paths, the strategy uses a hysteresis, which is implemented by increasing the total value of the current working face by x (currently 20) percent.

The MADM strategy also introduces the concept of a *Cost Estimator*: it sees cost as a variable attribute that can change according to user requirements and traffic behavior. The strategy uses this variable cost factor to automatically react to a traffic limit on an interface (like a provider-induced limit on a cellular network): the closer the traffic gets to the limit, the costlier it becomes to use this interface. This cost adjustment allows the strategy to offload unimportant traffic to lower cost (and possibly lower quality) interfaces.

To perform this traffic offloading, the strategy needs to set a few conventions. The current implementation specifies the default cost (c_{def}) of one interface as 100 and the maximal cost (c_{max}) as 1000. If an interface has a traffic limit, a function `adjustCost()` takes the current traffic load and increases the cost as follows:

$$c = max\{c_{def}, 1000 * \frac{t_{cur}}{t_{limit}}\} \tag{13}$$

Thus, if 90% of the traffic limit is reached, the cost will be set to 900. The application profile can use this information to define a cost limit; an application with a limit of $[600 - 800]$ discourages using interfaces that are over 60% of their limit and completely avoids those over 80%. Moreover, the strategy stops probing interfaces whose cost are above the upper limit to reduce unnecessary traffic load.

We have investigated the cost limit in a scenario with two consumer applications (Figure 13). The first application (App1) sends a constant stream of 100 packets per second (pps) and always chooses the lowest loss face; the second application (App2) sends 200 pps and uses the parameters maxloss=0[%]-40[%],maxcost=500-700, that is, it also prefers the lowest loss face, but adds a cost constraint. The topology is as follows: link 256 has 10% packet loss and a traffic limit of 10 megabyte; link 257 has 20% packet loss and no limit. This configuration roughly simulates a terminal with a high quality, but costly, cellular network and a low quality, but cheap, WiFi network.

We observed that App2 stops using the costly face after about 22 seconds when about 54% of the traffic limit is exhausted; it stops probing link 256 after about 36 seconds when *exactly* 70% of the traffic limit is reached. However, face 256 can still be probed or used by another application. Currently, every application is responsible for probing the interfaces it may want to use, which can create a large overhead when there are many concurrent applications. This probing overhead can be reduced by using a centralized probing instance. More general, interface probing leaves many questions for future work: how often to probe, which algorithms to use, and how fine-grained to aggregate probing measurements.

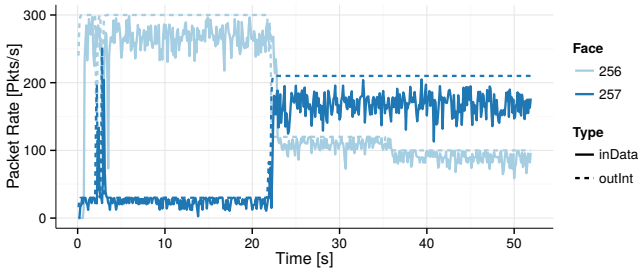


Figure 13: Behavior of the MADM Strategy in the Traffic Limit Scenario

Selective Parallel Strategy.

The Selective Parallel Strategy falls into the category of parallel strategies (Section 4.3) with the main target of real-time communication services. Like the Lowest Cost Strategy, it uses the parameters loss, delay, and bandwidth and chooses the interface with the lowest cost that satisfies all requirements. However, if one of the requirements is not met, it sends interest packets on multiple interfaces simultaneously until one face satisfies all requirements again. The number of interfaces that the strategy adds can be tuned based on the application needs. In the following evaluation, we use two alternatives: sending packets on the two best faces (*SP-Best2*) and flooding them on all available faces (*SP-Flood*).

We configure the paths of the topology from Figure 3 with delays of 200 ms, 400 ms, and 600 ms. The paths are probed every 10th packet (if the strategy uses probing) and the consumer application sends 300 pps without retransmissions. The packet loss of the paths is modeled as follows: every 2 seconds each path receives a random loss percentage between 0 and 30 %. The application profile is set to keep the total loss below 10%. We have compared a number of forwarding strategies according to their ability to do so; we compare two metrics over 30 measurement runs (see Figure 14): 1) the percentage of intervals of 3 seconds that are below 10% packet loss and 2) the amount of sent interests summed up over all faces.

We observed the following results: The best-route strategy (BestRoute) stays on one path, does not probe other paths, and thus sends exactly 300 pps. However, it has the highest packet loss of all strategies. The Lowest Cost Strategy (LCS) creates a much lower overall loss, since it changes the path as soon as a better one is available. It uses 10% more interest packets due to probing, but this amount was set manually and may be lower in practice.

To achieve an even better satisfaction of the loss requirement, one can use *redundant transmission*, that is, to send one interest packet simultaneously on multiple interfaces. The simplest of these strategies is to flood all packets among all available interfaces (Broadcast), but that method heavily increases overhead (here to 3 times the amount, since the topology uses 3 interfaces). The two versions of

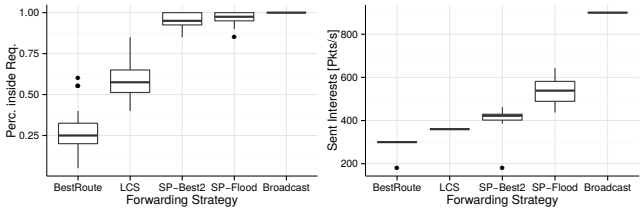


Figure 14: Performance of the Selective Parallel Strategy

the selective parallel strategy achieve nearly the same performance (as measured by the amount of intervals that satisfy the packet loss requirement), at a much lower overhead.

We use this evaluation to exemplify the possibilities of parallel forwarding. We recognize that it is a rather rough outline and future work has to expand on the details and use cases of this new adaptive forwarding method.

6. CONCLUSIONS

Today's mobile devices can transfer data by an unprecedented number of wireless technologies. To fully exploit their capabilities, one needs to move beyond the selection of single access networks towards more fine-grained control. We argue that NDN's content-oriented model is a great fit for this problem.

We have presented a system architecture that uses NDN's unique forwarding layer to improve end-user QoE and to reduce the terminals access cost and power consumption. Our prototype implementation illustrates the effects of specific mechanisms like interface probing, packet loss estimation, and the use of hysteresis thresholds. As shown in the evaluation, the system can adapt to fine-grained changes in link quality and adheres precisely to the application's cost tolerance.

Work still remains to be done in a number of areas. First, one can investigate which bandwidth estimation techniques can be adapted from IP networks. Moreover, the details of interface probing and of parallel forwarding strategies need to be worked out. A good strategy design in this area could strongly improve the QoE of real-time communication services in situations of suboptimal wireless link quality. Lastly, one should move from synthetic simulation scenarios to more realistic trace-driven scenarios and performance evaluation on real devices.

Acknowledgments

The authors want to thank Junxiao Shi and Alexander Afanasyev for helpful comments on the NFD implementation.

7. REFERENCES

[1] 3GPP. Services and service capabilities (release 12). ETSI TS 122 105 V12.1.0, Jan. 2015.

[2] A. J. Abu, B. Bensaou, and J. M. Wang. Interest packets retransmission in lossy ccn networks and its impact on network performance. In *Proceedings of ACM ICN*, 2014.

[3] G. Carofiglio, M. Gallo, L. Muscariello, and M. Papali. Multipath congestion control in content-centric networks. In *INFOCOM WKSHPS*, pages 363–368. IEEE, 2013.

[4] Project CCNx™. http://www.ccnx.org/.

[5] A. Detti, C. Pisa, and N. Blefari-Melazzi. Modeling multipath forwarding strategies in information centric networks. In *IEEE Global Internet Symposium*, 2015.

[6] A. Detti, M. Pomposini, N. Blefari-Melazzi, S. Salsano, and A. Bragagnini. Offloading cellular networks with icn: The case of video streaming. In *WoWMoM*, pages 1–3. IEEE, 2012.

[7] A. Detti, B. Ricci, and N. Blefari-Melazzi. Mobile peer-to-peer video streaming over information-centric networks. *Elsevier Computer Networks*, 81(0):272 – 288, 2015.

[8] Custom NFD strategy layer implementation. https://github.com/schneiderklaus.

[9] E. Gustafsson and A. Jonsson. Always best connected. *IEEE Wireless Communications*, 10(1):49–55, 2003.

[10] V. Jacobson, D. K. Smetters, N. H. Briggs, M. F. Plass, P. Stewart, J. D. Thornton, and R. L. Braynard. Voccn: voice-over content-centric networks. In *Proceedings of ACM ReArch*, 2009.

[11] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. H. Briggs, and R. L. Braynard. Networking named content. In *Proceedings of ACM CoNEXT*, 2009.

[12] M. Kassar, B. Kervella, and G. Pujolle. An overview of vertical handover decision strategies in heterogeneous wireless networks. *Elsevier Computer Communications*, 31(10):2607–2620, 2008.

[13] D.-h. Kim, J.-h. Kim, Y.-s. Kim, H.-s. Yoon, and I. Yeom. Mobility support in content centric networks. In *Proceedings of ACM SIGCOMM ICN Workshop*, 2012.

[14] S. Mastorakis, A. Afanasyev, I. Moiseenko, and L. Zhang. ndnSIM 2.0: A new version of the NDN simulator for NS-3. Technical Report NDN-0028, NDN, January 2015.

[15] R. Prasad, C. Dovrolis, M. Murray, and K. Claffy. Bandwidth estimation: metrics, measurement techniques, and tools. *Network, IEEE*, 17(6):27–35, 2003.

[16] R. Ravindran, S. Lo, X. Zhang, and G. Wang. Supporting seamless mobility in named data networking. In *IEEE ICC*, 2012.

[17] K. M. Schneider, K. Mast, and U. R. Krieger. CCN forwarding strategies for multihomed mobile terminals. In *Proceedings of NetSys*, 2015.

[18] C.-L. Tsao and R. Sivakumar. On effectively exploiting multiple wireless interfaces in mobile hosts. In *Proceedings of ACM CoNEXT*, 2009.

[19] L. Wang and G.-S. Kuo. Mathematical modeling for network selection in heterogeneous wireless networks – a tutorial. *IEEE Communications Surveys & Tutorials*, 15(1):271–292, 2013.

[20] L. Wang, O. Waltari, and J. Kangasharju. Mobiccn: Mobility support with greedy routing in content-centric networks. In *IEEE GLOBECOM*, 2013.

[21] X. Yan, Y. A. Şekercioğlu, and S. Narayanan. A survey of vertical handover decision algorithms in fourth generation heterogeneous wireless networks. *Elsevier Computer Networks*, 54(11):1848–1863, 2010.

[22] C. Yi, J. Abraham, A. Afanasyev, L. Wang, B. Zhang, and L. Zhang. On the role of routing in named data networking. In *Proceedings of ACM ICN*, 2014.

[23] C. Yi, A. Afanasyev, L. Wang, B. Zhang, and L. Zhang. Adaptive forwarding in named data networking. *SIGCOMM Comput. Commun. Rev.*, 42(3):62–67, June 2012.

[24] Z. Zhu, A. Afanasyev, and L. Zhang. A new perspective on mobility support. *Named-Data Networking Project, Tech. Rep*, 2013.

[25] Z. Zhu, S. Wang, X. Yang, V. Jacobson, and L. Zhang. Act: audio conference tool over named data networking. In *Proceedings of ACM SIGCOMM ICN Workshop*, 2011.