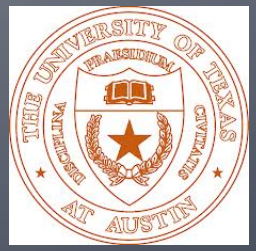
A Case Study of Cross-System Porting in Forked Software Projects

Baishakhi Ray and Miryung Kim
The University of Texas at Austin



Motivation

- Software forking has become popular.
- Developers may need to port similar feature additions and bug-fixes across the projects.
- The characteristics of repeated work required to maintain forked projects is yet unknown.

Study Findings

- Cross system patch porting happens periodically.
- Porting practice heavily depends on core developers doing their porting job on time.
- Ported changes are less defect-prone than non-ported changes.
- Ported changes are localized.

Outline

- Related Work
- Study Subjects
- Repertoire Approach
- Research Questions & Results
- Conclusions

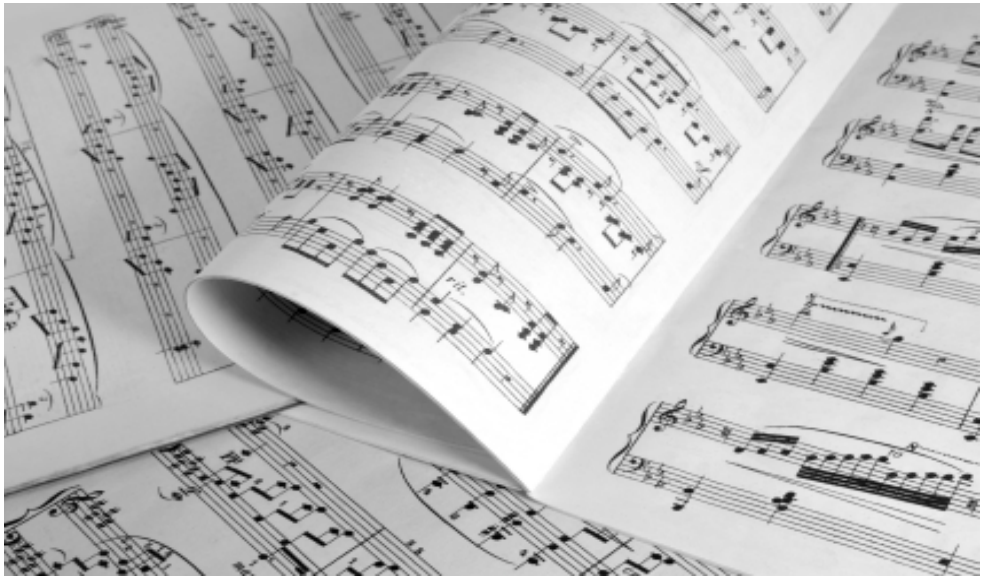
Related Work

- Code clone analysis [Kamiya et al., Jiang et al., Baker et al.].
  - Detect only duplicate code
  - Cannot detect repeated work involved in cross-system porting
- Case studies on the BSD product family
  - Focus on cross-system communications [Canfora et al.]
  - Analyze copy-right implications of code flow [German et al.].
- Studies on recurring bug fixes
  - Investigate only individual projects as opposed to a product family [Nguyen et al.].

Study Subjects

Projects	KLOC	Releases	Authors	Years
FreeBSD	359 to 4479	54 (R1.0 - R8.2)	405	18
NetBSD	859 to 4463	14 (R1.0 - R5.1)	331	18
OpenBSD	297 to 2097	30 (R1.1 - R5.0)	264	16

Repertoire (FSE'12 tool-demo)



- Input: a sequence of *diff* based program patches from forked projects.
- Output: ported edits among the patches.
- Repertoire compares patches to identify similar contents and edit operations.

Step 1: Identify cloned regions using CCFinderX [Kamiya et al.]

Patch1 (Jan '10)	Patch2 (Mar '10)
**** Old ****	**** Old ****
X1 for(i=0;i<MAX;i++){	Y1 for(j=0;j<MAX;j++) {
X2 - x = array[i]+x;	Y2 q = p + q;
X3 - y = foo(x);	Y3 - q = array[j]+p;
X4 - x = x-y;	Y4 - p = foo1(q);
X5 }	Y5 }
**** New ****	**** New ****
X6 for(i=0;i<MAX;i++) {	Y6 for(j=0;j<MAX;j++) {
X7 + y = x+y;	Y7 q = p + q;
X8 + x = array[i]+x;	Y8 + q = array[j] + q;
X9 + y = foo(x,y);	Y9 + p = foo1(p,q);
X10 }	Y10 }

Step 2: Match edit operations of cloned regions

Patch1 (Jan '10)	Patch2 (Mar '10)
**** Old ****	**** Old ****
X1 for(i=0;i<MAX;i++){	Y1 for(j=0;j<MAX;j++) {
X2 - x = array[i]+x;	Y2 q = p + q;
X3 - y = foo(x);	Y3 - q = array[j]+p;
X4 - x = x-y;	Y4 - p = foo1(q);
X5 }	Y5 }
**** New ****	**** New ****
X6 for(i=0;i<MAX;i++) {	Y6 for(j=0;j<MAX;j++) {
X7 + y = x+y;	Y7 q = p + q;
X8 + x = array[i]+x;	Y8 + q = array[j] + q;
X9 + y = foo(x,y);	Y9 + p = foo1(p,q);
X10 }	Y10 }

Step 2: Match edit operations of cloned regions

Patch1 (Jan '10)	Patch2 (Mar '10)
**** Old ****	**** Old ****
X1 for(i=0;i<MAX;i++){	Y1 for(j=0;j<MAX;j++) {
X2 - x = array[i]+x;	Y2 q = p + q;
X3 - y = foo(x);	Y3 - q = array[j]+p;
X4 - x = x-y;	Y4 - p = foo1(q);
X5 }	Y5 }
**** New ****	**** New ****
X6 for(i=0;i<MAX;i++) {	Y6 for(j=0;j<MAX;j++) {
X7 + y = x+y;	Y7 q = p + q;
X8 + x = array[i]+x;	Y8 + q = array[j] + q;
X9 + y = foo(x,y);	Y9 + p = foo1(p,q);
X10 }	Y10 }

Ported edits

Step 3: Disambiguate source and destination of ported edit

Patch1 (Jan '10)	Patch2 (Mar '10)
**** Old ****	**** Old ****
X1 for(i=0;i<MAX;i++){	Y1 for(j=0;j<MAX;j++) {
X2 - x = array[i]+x;	Y2 q = p + q;
X3 - y = foo(x);	Y3 - q = array[j]+p;
X4 - x = x-y;	Y4 - p = foo1(q);
X5 }	Y5 }
**** New ****	**** New ****
X6 for(i=0;i<MAX;i++) {	Y6 for(j=0;j<MAX;j++) {
X7 + y = x+y;	Y7 q = p + q;
X8 + x = array[i]+x;	Y8 + q = array[j] + q;
X9 + y = foo(x,y);	Y9 + p = foo1(p,q);
X10 }	Y10 }

Patch Porting Example from FreeBSD to NestBSD

FreeBSD Patch (bin/cp/cp.c : rev 1.3)
Date:1994/12/30
Author: bde

Change Log:
...
Be more careful about concatenating pathnames: don't check that the pathname fits until prefixes have been discarded
...

```
 p = &curr->fts_path[base];
 nlen = curr->fts_pathlen - base;
!  target_mid = to.target_end;
!  if (*p != '/' && target_mid[-1] != '/')
!  *target_mid++ = '/';
! *target_mid = 0;
```

...

NetBSD Patch (bin/cp/cp.c : rev 1.40)
Date: 2005/11/16
Author: christos

Change Log:
- Better detect pathname overflow (from FreeBSD)
- Change destination normal file detection to match with FreeBSD
...

```
  p = &curr->fts_path[base];
  nlen = curr->fts_pathlen - base;
! target_mid = to.target_end;
!  if (*p != '/' && target_mid[-1] != '/')
!  *target_mid++ = '/';
!  *target_mid = 0;
```

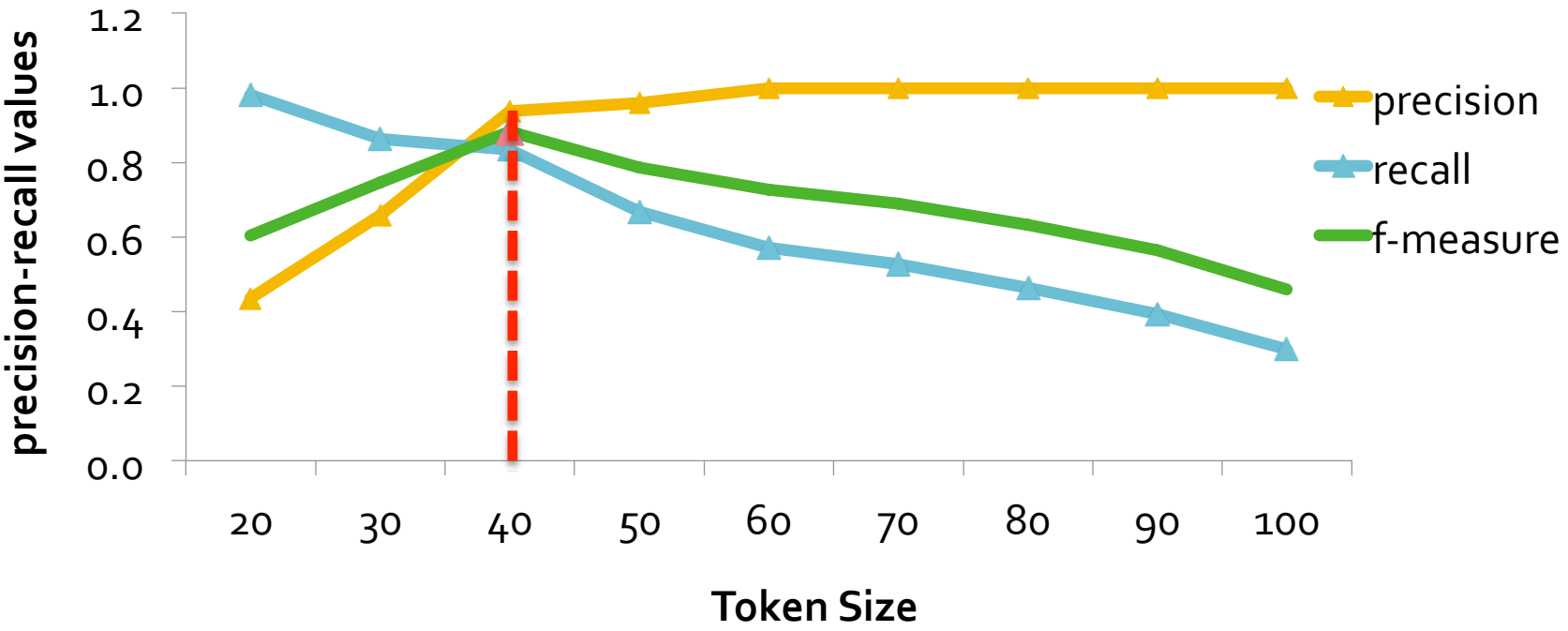
...

Accuracy Measurement

- We manually constructed a ground truth set of edits ported from NetBSD to OpenBSD releases 4.4 and 4.5.

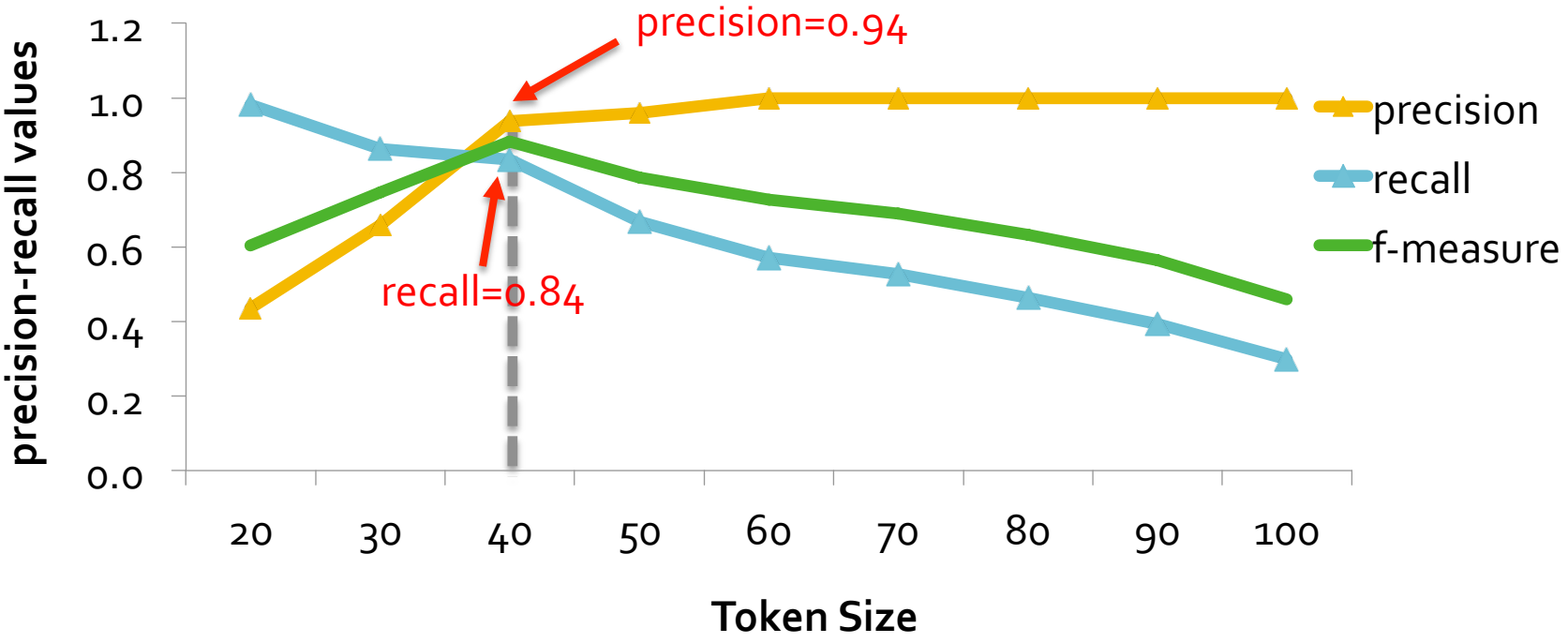
- We evaluate with Repertoire's output against the ground truth set, while varying the token size threshold for CCFinderX [Kamiya et al.].

Accuracy Measurement



- Precision: 94%, Recall: 84%
- Token threshold: 40

Accuracy Measurement



- Precision: 94%, Recall: 84%
- Token threshold: 40

Outline

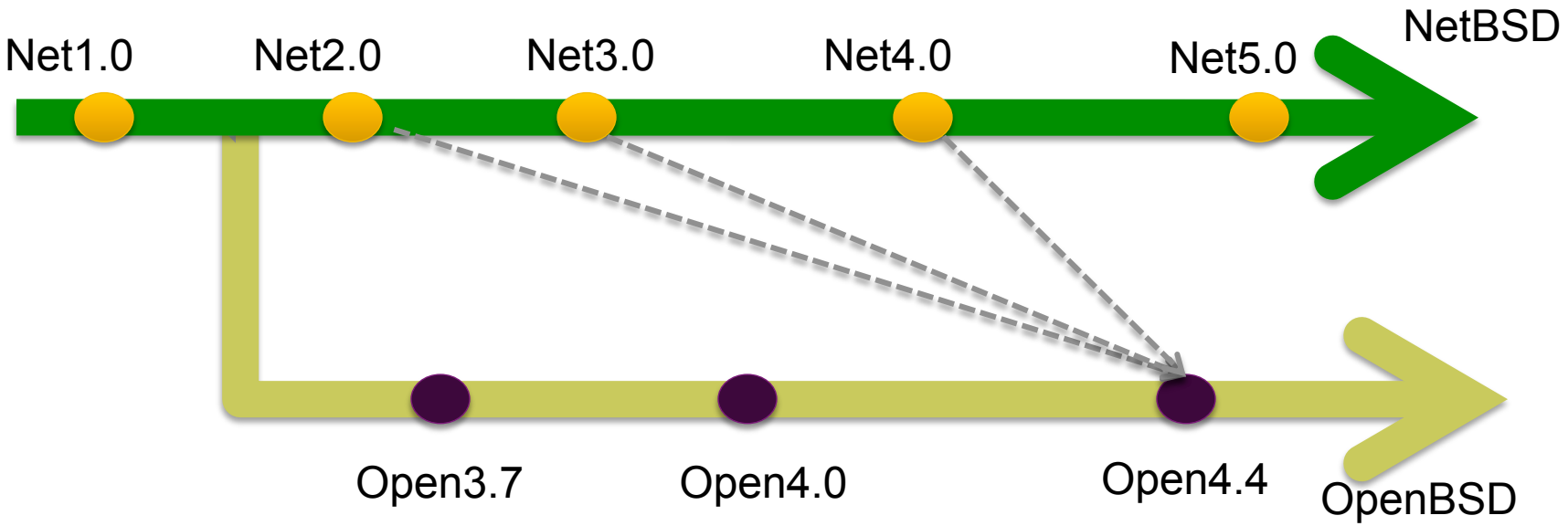
- Related Work
- Study Subjects
- Repertoire Approach
- **Research Questions & Results**
- Conclusions

Research Questions

- Q1: What is the extent of changes ported from other projects?
- Q2: Are ported changes more defect-prone than non-ported changes?
- Q3: How many developers are involved in porting patches from other projects?
- Q4: How long does it take for a patch to propagate to different projects?
- Q5: Where is the porting effort focused on?

Q1: What is the extent of changes ported from other projects?

- Methodology
  - Compare program patches at release granularity.



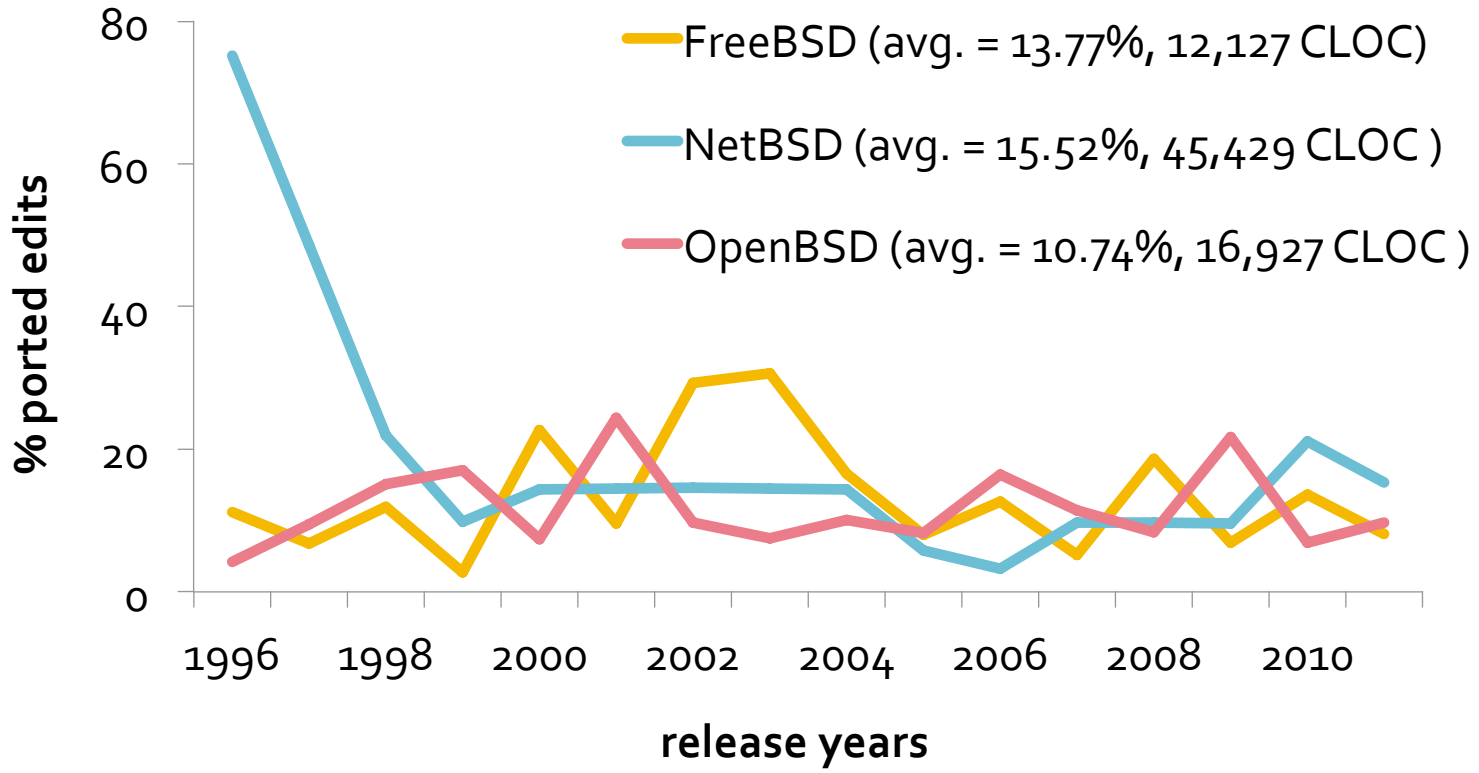
Q1: What is the extent of changes ported from other projects?

- Methodology
  - Compare program patches at release granularity.
  - Identify ported lines.
  - Compute porting rate.

$$avg\_porting\_rate = \frac{\sum_{releases} ported\_edits}{\sum_{releases} total\_edits}$$

  - Example: If a patch contains 10 lines of total edits, where 5 of them are ported from another project, porting rate is 50% on average.

Q1: What is the extent of changes ported from other projects?



Porting is significant in the BSD family evolution, and it is not necessarily decreasing over time.

Q2:Are ported edits more defect prone than non-ported edits?

- Methodology
  - Measure ported and non-ported lines using Repertoire.
  - Measure Spearman rank correlation between the number of bug fixes [Mockus and Votta] and ported and non-ported lines respectively, at file granularity.

Q2:Are ported edits more defect prone than non-ported edits?

	CLOC	Ported CLOC	Non-ported CLOC
FreeBSD	0.26	0.15	0.25
NetBSD	0.41	0.36	0.42
OpenBSD	0.37	0.32	0.38

Files with ported edits are less defect-prone than the files with non-ported edits

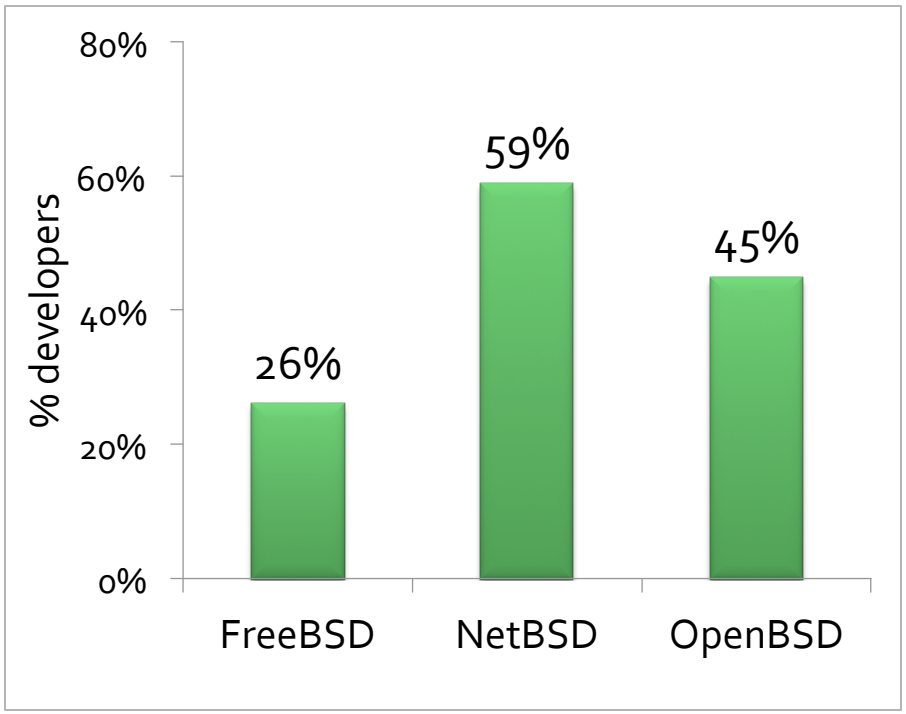
Q3: How many developers are involved in porting patches from other projects?

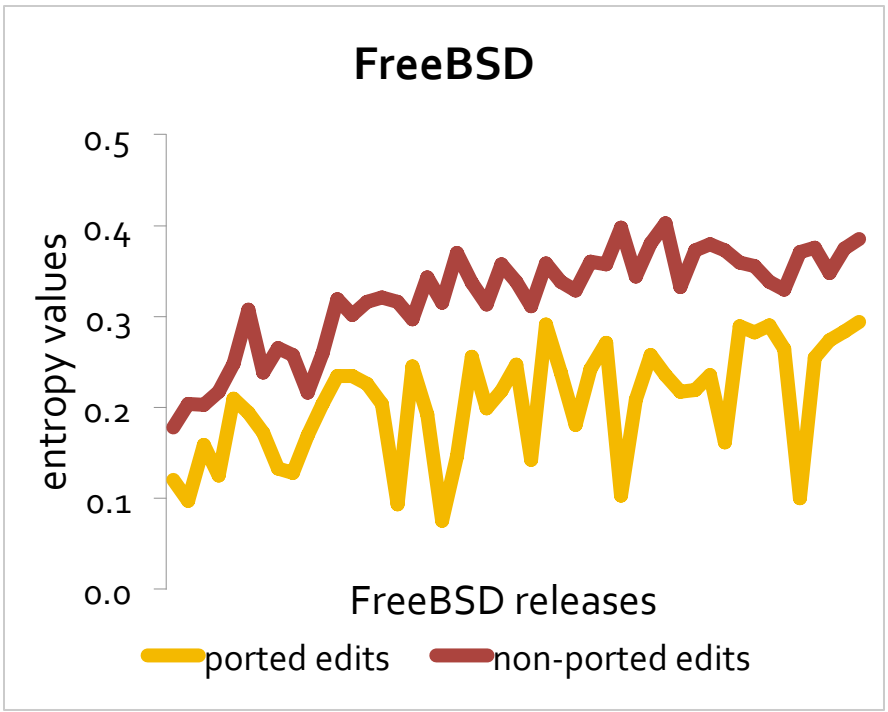
- Methodology
  - Measure the percentage of developers involved in porting.
  - Measure porting workload distribution by calculating normalized entropy score of developers' contribution [Hassan et al].

$$normalized\_entropy = -\sum_{i=i}^{n} p_i * \log_n(p_i)$$

  - If entropy is high, the workload is more equally distributed among the contributors.

Q3: How many developers are involved in porting patches from other projects?

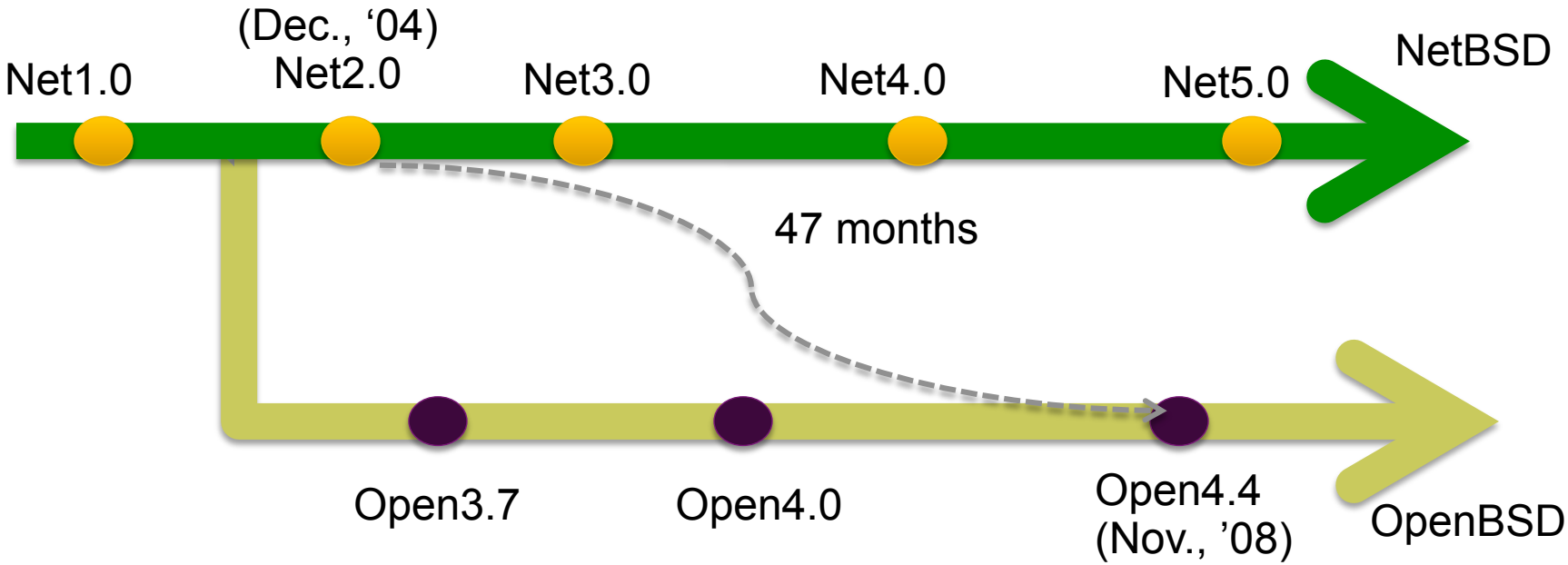




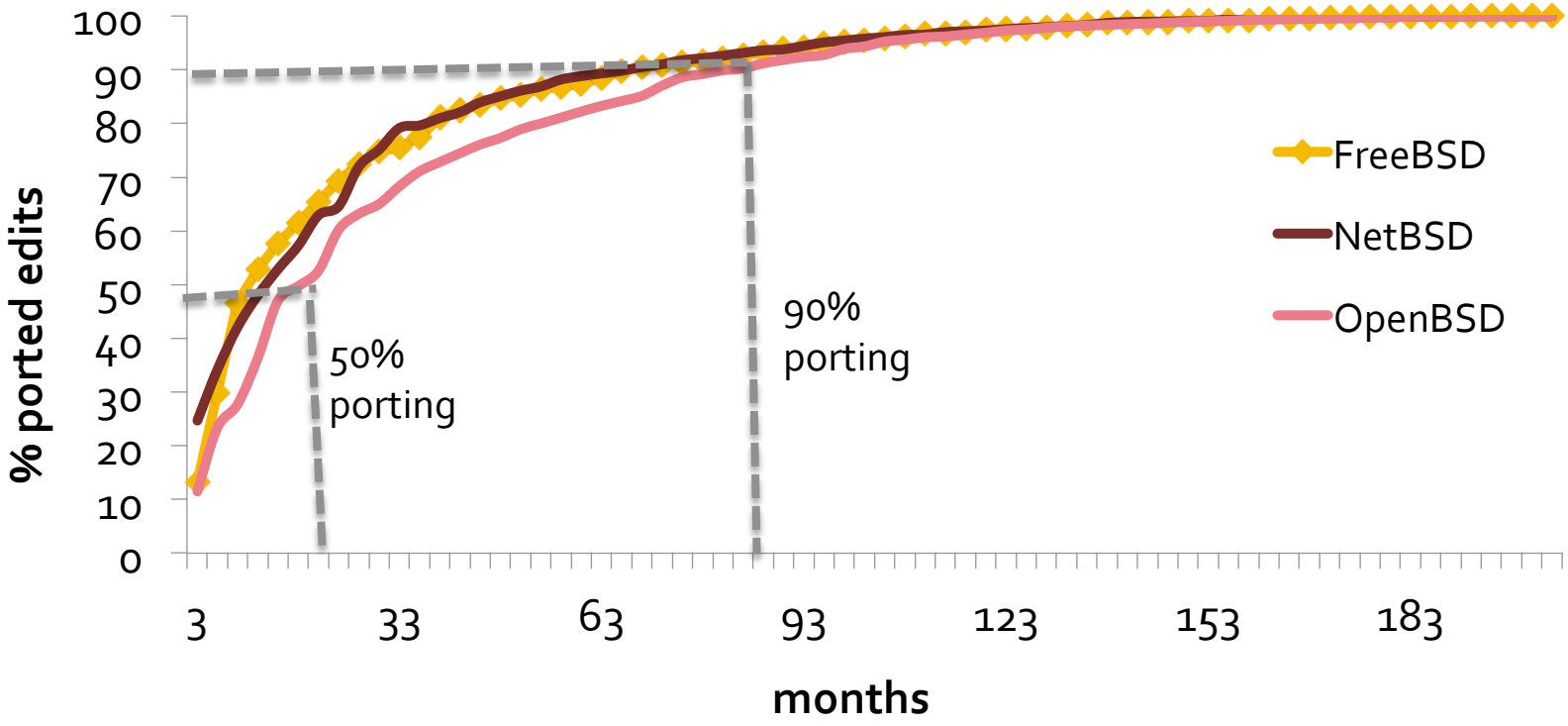
A significant portion of active committers port changes, but some do more porting work than others.

Q4: How long does it take for a patch to propagate to different projects?

- Methodology
  - A patch propagation latency = target patch release date – source patch release date.



Q4: How long does it take for a patch to propagate to different projects?

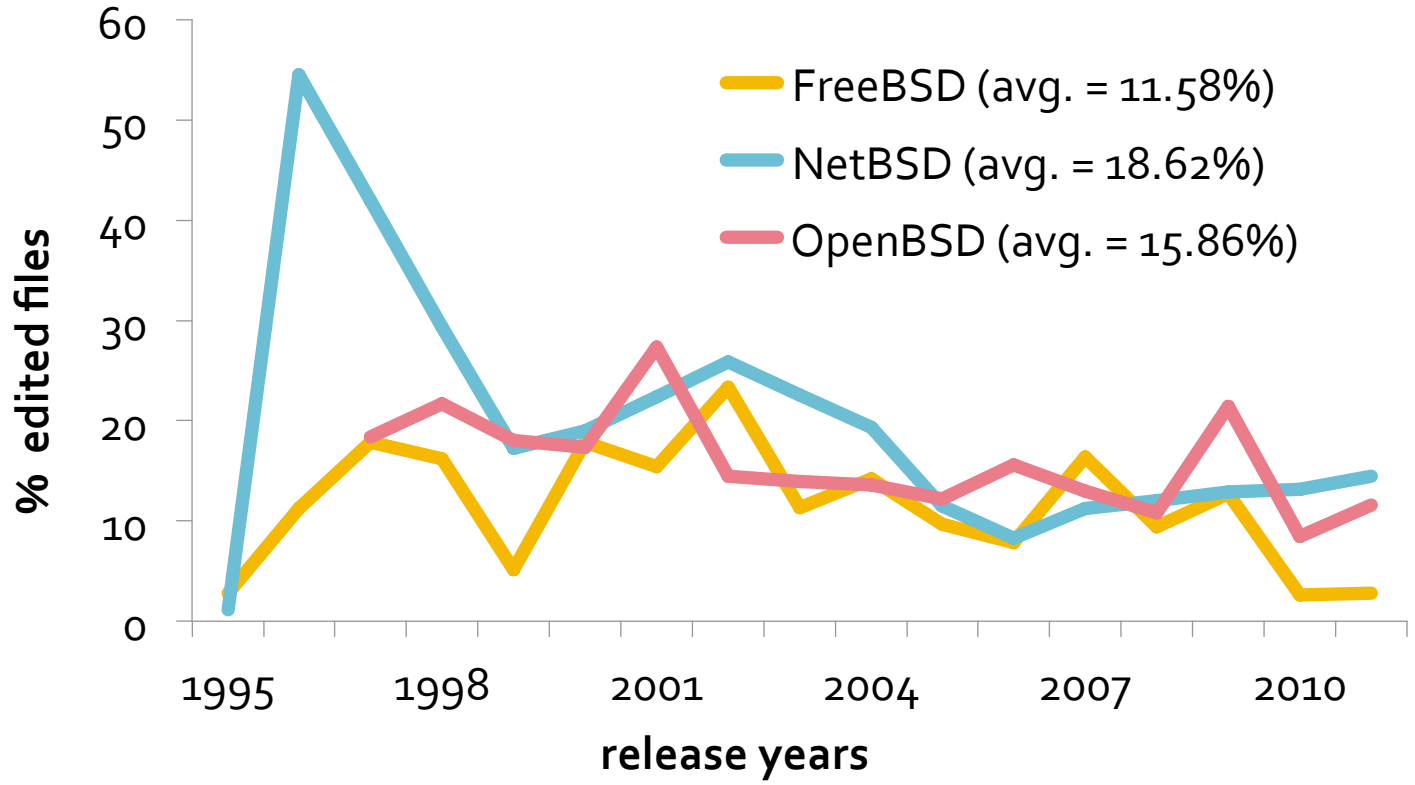


While most ported changes migrate to peer projects in a relatively short amount of time, some changes take a very long time to propagate to other projects.

Q5: Where is the porting effort focused on?

- Methodology
  - Measure the file level distribution of ported edits in each BSD project.
  - Consider a file is affected by porting in the i^{th} release, if it is modified by at least one ported edit since its previous release.

Q5: Where is the porting effort focused on?



Ported changes affect about 12% to 19% of modified files and porting effort is concentrated on specific parts of the BSD codebase.

Q5: Where is the porting effort focused on?

- Top 4 directories with the largest amount of ported changes.

Rank	FreeBSD		NetBSD		OpenBSD	
1	src/crypto/openssl	21.54%	src/sys/arch	20.34%	src/sys/dev	24.57%
2	src/crypto/openssh	13.98%	src/sys/dev	19.96%	src/lib/libssl	16.36%
3	src/crypto/heimdal	13.31%	src/crypto/dist	10.61%	src/sys/arch	11.16%
4	src/sys/dev	8.95%	src/gnu/dist	4.54%	src/usr.sbin/ppp	6.27%

Summary

- Repertoire analyzes cross-system porting in temporal, spatial and developer dimension.
  - The repeated maintenance work is significant.
  - Ported changes are more reliable than non-ported changes.
  - Cross-system porting in the BSDs heavily depends on developers doing their porting job on time.

Summary

- Calls for automated approaches for cross-system porting [Meng et al.,Anderson et al.]
- Calls for tools to notify developers of potential collateral evolution and cross-system change impact analysis

Acknowledgment

- We thank Jihun Park for gathering the bug history data for FreeBSD, NetBSD, and OpenBSD projects.

- This work was in part supported by National Science Foundation under the grants CAREER-1117902, CCF-1149391, and CCF-1043810 and Microsoft SEIF award.

- Data sets and Repertoire tool are available for public. http://dolphin.ece.utexas.edu/Repertoire.html

A Case Study of Cross-System Porting in Forked Software Projects

http://dolphin.ece.utexas.edu/Repertoire.html

Baishakhi Ray and Miryung Kim
The University of Texas at Austin